NOVA SCHOOL OF SCIENCE & TECHNOLOGY
DEPARTAMENTO DE INFORMÁTICA

Arquitetura de Computadores 2024/25

TPC 3

Deadline for delivery: May 20th, 2025; 6:00pm

This homework consists of a programming exercise to be carried out in a group of no more than two students. You can ask general questions to colleagues, but the solution and writing of code must be strictly carried out by group members. Cases of plagiarism will be punished in accordance with current regulations.
The delivery method will be through Mooshak, using the identifiers of the previous work, unless the group has been changed. More detailed information will be sent via a CLIP message.

Encrypt and decrypt with the Tiny Encryption Algorithm (TEA)

In this exercise, you will impement two programs that allow the encryption and decryption of a file using the TEA (Tiny Encryption Algorithm) method. The TEA method uses a 128-bit key and the file is encrypted/decrypted by dividing it into 64-bit blocks. The algorithms for encrypting and decrypting a block are as follows:

- the data to be encrypted/decrypted is in the text vector with two 32-bit integers
- the key is in the vector k with four 32-bit integers

```
void encrypt(unsigned int text[], unsigned int k[]) {
  unsigned int y = text[0], z = text[1];
  unsigned int delta = 0x9e3779b9, sum = 0;
  int n;
  for (n= 0; n < 32; n++)
    {
      sum += delta;
      y += ((z << 4) + k[0]) ^ (z+sum) ^ ((z >> 5) + k[1]);
      z += ((y << 4) + k[2]) ^ (y+sum) ^ ((y >> 5) + k[3]);
    }
  text[0] = y;  text[1] = z;
}

void decrypt(unsigned int text[], unsigned int k[]) {
  unsigned int y = text[0], z = text[1];
  unsigned int delta = 0x9e3779b9, sum = delta << 5;
  int n;
  for (n= 0; n < 32; n++)
    {
      z -= ((y << 4) + k[2]) ^ (y + sum) ^ ((y >> 5) + k[3]);
      y -= ((z << 4) + k[0]) ^ (z + sum) ^ ((z >> 5) + k[1]);
      sum -= delta;
    }
  text[0] = y; text[1] = z;
}
```

The following *Makefile* describes how three executable files are built.

```
CC=gcc
CFLAGS=-Wall -std=c99 -g
ASFLAGS=-gstabs

all: genKey crypt-tea decrypt-tea

genKey: genKey.c.c
    $(CC) -Wall -g -o genKey genKey.c

cryptTEA: cryptTEA.c tea.o
    $(CC) $(CFLAGS) -o cryptTEA cryptTEA.c tea.o

decryptTEA: decryptTEA.c tea.o
    $(CC) $(CFLAGS) -o cryptTEA decryptTEA.c tea.o

tea.o:tea.s
    as $(ASFLAGS) -o tea.o tea.s

clean:
    rm -f *.o genKey cryptTEA decryptTEA
```

Now, the operations of the three programs are described.

genKey

The program is invoked from the command line through:

```
./genkey file_name_to_store_key number_of_bytes_of_key
```

Gets a key with the number of bytes specified in the 2nd argument; the key is stored in binary in the file whose name is the 1st argument. In the case of the TEA algorithm, the key has 16 bytes (128 bits).

cryptTEA

The program is invoked from the command line through:

```
./cryptTEA file_with_key  file_plain file_ciphered
```

Uses the key in the file whose name is the 1st argument to encrypt the file whose name is the 2nd argument with the TEA algorithm; the encrypted file is saved with the name that is the 3rd argument.

decryptTEA

The program is invoked from the command line through: :

```
/decryptTEA file_with_key  file_cyphered file_plain
```

Uses the key in the file whose name is the 1st argument to decode the file whose name is the 2nd argument with the TEA algorithm; the file with the clear content is saved with the name that is the 3rd argument.

Work to perform

The files *Makefile*, `genKey.c`, and `decryptTEA.c` are supplied in source form and should not be modified. You should develop and test the following source files:

- `tea.s` - This file contains code of the *crypt* and *decrypt* functions, specified above, in x86-64 assembly with ATT syntax. The input parameters are also as indicated above.
- `cryptTEA.c` – Make a copy of the *decryptTEA.c* file into a *cryptTEA.c* file. Modify it so that it behaves according to the above specification. One important change is that for the algorithm to work, the input block must be 64 bits (8 bytes) long; so if the file has a length that is not a multiple of 8, you need to add 0 bytes to the end, so that the file length (in bytes) becomes a multiple of 8. This operation is called *padding*.

To test your implementation, run the following commands for different file types:

```
./genKey key 16
./cryptTEA key xxx yyy
./decryptTEA key yyy zzz
diff xxx zzz
```

expecting that no differences appear. Note that xxx and zzz can be different if padding was used.

Delivery

Submission must be made via the Mooshak system, as described above; it must be done within the deadline, by sending a zip file. This file must contain only the files *cryptTEA.c* and *tea.s*.

The programs will be compiled using the provided *Makefile*.